SPARQL Query Containment under RDFS Entailment Regime

Melisachew Wudagae Chekol, Jérôme Euzenat, Pierre Genevès, Nabil Layaïda

SPARQL Query Containment under RDFS Entailment Regime

Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, Nabil Layaïda

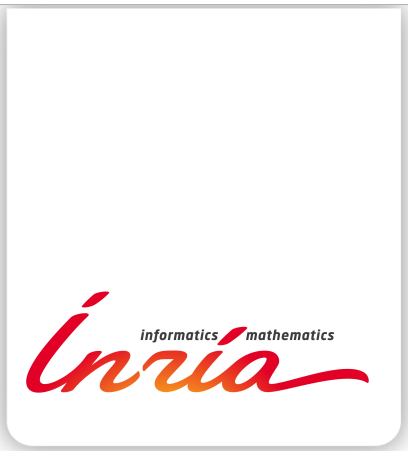
RESEARCH REPORT

N° 7942

April 2012

Project-Teams EXMO and WAM

ISSN 0249-6399 ISRN INRIA/RR--7942--FR+ENG



SPARQL Query Containment under RDFS Entailment Regime

Melisachew Wudage Chekol*, Jérôme Euzenat*, Pierre Genevès†, Nabil Layaïda*

Project-Teams EXMO and WAM

Research Report n° 7942 — April 2012 — 22 pages

Abstract: The problem of SPARQL query containment is defined as determining if the result of one query is included in the result of another for any RDF graph. Query containment is important in many areas, including information integration, query optimization, and reasoning about Entity-Relationship diagrams. We encode this problem into an expressive logic called μ-calculus: where RDF graphs become transition systems, queries and schema axioms become formulas. Thus, the containment problem is reduced to formula satisfiability test. Beyond the logic's expressive power, satisfiability solvers are available for it. Hence, this study allows to exploit these advantages.

Key-words: Query containment, SPARQL, entailment regime, ontologies, RDF

* INRIA
† CNRS

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Inclusion de requêtes sur les axiom RDFS

Résumé :

Mots-clés : inclusion de requêtes, SPARQL, entailment regime,ontologies, RDF

1 Introduction

SPARQL is a W3C recommended query language for RDF. The language is being extended with different entailment regimes and regular path expressions[1]. The semantics of SPARQL relies on the definition of basic graph pattern matching that is built on top of RDF simple entailment [14]. However, it may be desirable to use SPARQL to query triples entailed from subclass, subproperty, range, domain, and other relations which can be represented using RDF schema. The SPARQL specification defines the results of queries based on RDF simple entailment. The specification also presents a general parametrized definition of graph pattern matching that can be expanded to other entailments beyond simple entailment. Query answering under the RDFS entailment regime can be achieved via: (1) materialization (computing the deductive closure of the queried graph) [13], (2) rewriting the queries using the schema, and (3) hybrid (combining materialization and query rewriting). We use a technique based on the approaches (1) and (2) to study the problem of SPARQL query containment under the RDFS entailment regime.

Studies on the translation of SPARQL into relational algebra and SQL [11, 9] indicate a close connection between SPARQL and relational algebra in terms of expressiveness. In [20], a translation of SPARQL queries into a datalog fragment (non-recursive datalog with negation) that is known to be equally expressive as relational algebra was presented. This translation makes the close connection between SPARQL and rule-based languages explicit and shows that RA is at least as expressive as SPARQL. Tackling the opposite direction, it was recently shown in [2] that SPARQL is relationally complete, by providing a translation of the above-mentioned datalog fragment into SPARQL. As argued in [2], the results from [20] and [2] taken together imply that SPARQL has exactly the same expressive power as relational algebra. From early results on query containment in relational algebra and first-order logic, one can learn that containment in relation algebra is undecidable (contrary to the results in [10]). Therefore, containment of SPARQL queries is also undecidable. Hence, we consider a fragment of (P)SPARQL containing only conjunction and disjunction for this study.

Query containment is defined as determining if the result of one query is included in the result of another one for any RDF graph. It has been a central point of research due to its vital role in query optimization, information integration and reasoning about Entity-Relationship diagrams [17, 8]. In [5], a double exponential upper bound is proved for containment of union of conjunctive queries (UCQs) under expressive description logic constraints. Beyond UCQs, containment of (two-way) regular path queries (2RPQs) have been studied extensively [7, 3]. These languages are used to query graph databases and containment has been shown to be PSPACE-complete and EXPTIME-hard under the presence of functionality constraints [7]. On the other hand, the containment of conjunctive 2RPQs is EXPSPACE-complete, this bound jumps to 2EXPTIME when considered under expressive description logic (DL) constraints [6]. In fact, this problem has already been implicitly addressed in [5] when $\mathcal{DLR}$ (DLs with n-ary relations) constraints are used. More recently, Path SPARQL (PSPARQL [1]) query containment has been studied in [10] where a double exponential upper bound is established. In this work, we consider the same approach as [10] and prove that containment of PSPARQL queries under RDF schema axioms has a double exponential upper bound. However, it is exponential if the query on the right hand side has a tree structure (cf. for example, [5]). Further, paths are being included in the new version of SPARQL, thus this work can be used to test containment of path SPARQL queries under the RDFS entailment regime.

To study containment, we apply an approach which has already been successfully applied for XPath [12]. SPARQL is interpreted over graphs, hence we encode it in a graph logic, specifically the alternation-free fragment of the μ-calculus [18] with converse and nominals [22] interpreted

[1] `http://www.w3.org/TR/sparql11-query/`

over labeled transition systems. We show that this logic is powerful enough to deal with query containment for union of conjunctive SPARQL queries under the RDFS entailment regime. Furthermore, this logic admits exponential time decision procedures that is implemented in practice [22, 23, 12]. Hence, our approach opens a way to take advantage of these implementations. We introduce a translation of RDF graphs into transition systems and SPARQL queries and RDF schema into μ-calculus formulae. Then, we show how query containment in SPARQL under RDFS entailment can be reduced to unsatisfiability in the μ-calculus.

In summary, the contribution of this work is fourfold: (1) we formulate the problem of query containment under the RDFS entailment regime in three different ways, (2) since paths are included in the new version of SPARQL, this work can be used to determine containment of path queries (under RDF schema as well), (3) we show how to extend the schema language to the description logic $\mathcal{SH}$ (DL with role transitivity $\mathcal{S}$ and role hierarchy $\mathcal{H}$), and (4) we prove a double exponential upper bound for containment.

Outline: after presenting RDF(S) and SPARQL (§2), we show how to translate RDF graphs into transition systems (§3) and SPARQL queries into μ-calculus formulas (§4). Therefore, query containment in SPARQL under RDFS entailment is reduced to unsatisfiability test in μ-calculus (§5). Finally, we present the complexity of the problem (§5.4) along with a summary of concluding remarks (§6).

2 Preliminaries

This section introduces the foundations of RDF(S), SPARQL, and μ-calculus.

2.1 RDF(S)

RDF is a language used to express structured information on the Web as graphs. We present a compact formalization of RDF [14]. Let U, B, and L be three disjoint infinite sets denoting the set of URIs (identifying a resource), blank nodes (denoting an unidentified resource) and literals (a character string or some other type of data) respectively. We abbreviate any union of these sets as for instance, UBL $= U \cup B \cup L$. A triple of the form $(s, p, o) \in \text{UB} \times U \times \text{UBL}$ is called an *RDF triple*. s is the *subject*, p is the *predicate*, and o is the *object* of the triple. Each triple can be thought of as an edge between the subject and the object labelled by the predicate, hence a set of RDF triples is often referred to as an *RDF graph*. RDF has a model theoretic semantics [14].

Example 1 (RDF Graph)**.** *Consider the following RDF graph (all identifiers correspond to URIs and _:b is a blank node):*

$$G = \{(john, childOf, mary), (childOf, \texttt{sp}, ancestor), (\_:b, hasFather, john), \\ (ancestor, \texttt{dom}, Person), (ancestor, \texttt{range}, Person)\}$$

RDFS (short for RDF Schema) may be considered as a simple ontology language expressing subsumption relations between classes or properties [14]. Technically, this is an RDF vocabulary used for expressing axioms constraining the interpretation of graphs. The RDFS vocabulary and its semantics are given in [14]. There, inference rules (shown in equation (1)–(9)) are given which allow to deduce or infer new triples using the schema and RDF graph.

- Subclass (sc)

$$\frac{(a, \texttt{sc}, b)\ (b, \texttt{sc}, c)}{(a, \texttt{sc}, c)} \qquad \frac{(a, \texttt{sc}, b)\ (x, \texttt{type}, a)}{(x, \texttt{type}, b)} \tag{1}$$

- Subproperty (sp)

$$\frac{(a, \texttt{sp}, b)\ (b, \texttt{sp}, c)}{(a, \texttt{sp}, c)} \qquad \frac{(a, \texttt{sp}, b)\ (x, a, y)}{(x, b, y)} \tag{2}$$

- Typing (dom, range)

$$\frac{(a, \texttt{dom}, b)\ (x, a, y)}{(x, \texttt{type}, b)} \qquad \frac{(a, \texttt{range}, b)\ (x, a, y)}{(y, \texttt{type}, b)} \tag{3}$$

- Implicit Typing

$$\frac{(a, \texttt{dom}, \texttt{b})\ (\texttt{c}, \texttt{sp}, \texttt{a})\ (\texttt{x}, \texttt{c}, \texttt{y})}{(x, \texttt{type}, \texttt{b})} \qquad \frac{(a, \texttt{range}, \texttt{b})\ (\texttt{c}, \texttt{sp}, \texttt{a})\ (\texttt{x}, \texttt{c}, \texttt{y})}{(y, \texttt{type}, \texttt{b})} \tag{4}$$

- Subclass reflexivity

$$\frac{(a, \texttt{type}, \texttt{Class})}{(a, \texttt{sc}, \texttt{a})} \qquad \frac{(a, \texttt{sc}, \texttt{b})}{(a, \texttt{sc}, \texttt{a})\ (\texttt{b}, \texttt{sc}, \texttt{b})} \tag{5}$$

- Subproperty reflexivity

$$\frac{(a, \texttt{type}, \texttt{Property})}{(a, \texttt{sp}, \texttt{a})} \qquad \frac{(x, a, y)}{(a, \texttt{sp}, \texttt{a})} \tag{6}$$

- Resource

$$\frac{(a, b, c)}{(a, \texttt{type}, \texttt{Resource})} \qquad \frac{(a, b, c)}{(c, \texttt{type}, \texttt{Resource})} \qquad \frac{(a, \texttt{type}, \texttt{Class})}{(a, \texttt{sc}, \texttt{Resource})} \tag{7}$$

- Property

$$\frac{(a, b, c)}{(b, \texttt{type}, \texttt{Property})} \tag{8}$$

- Class

$$\frac{(a, b, c)}{(a, \texttt{type}, \texttt{Class})} \qquad \frac{(a, \texttt{type}, \texttt{c})}{(c, \texttt{type}, \texttt{Class})} \tag{9}$$

Example 2. *Using the inference rules, we can be infer the triples {(john,* ***type****,Person), (mary,* ***type****,Person), (john, ancestor,mary)}. Hence, the deductive closure of graph G in Example 1 contains:*

$$\begin{aligned} cl(G) = \{&(john, childOf, mary), (childOf, \textbf{\textit{sp}}, ancestor), (\_:b, hasFather, john), \\ &(john, \textbf{\textit{type}}, Person), (mary, \textbf{\textit{type}}, Person), (john, ancestor, mary), \\ &(ancestor, \textbf{\textit{dom}}, Person)\} \end{aligned}$$

In the next section, we present the query language SPARQL which is used to query RDF graphs.

2.2 SPARQL

SPARQL is a W3C recommended query language for RDF [21]. PSPARQL (Path SPARQL) extends SPARQL with regular expression patterns [1]. PSPARQL overcomes the limitation of the current version of SPARQL which is the inability to express path queries.

Before presenting the syntax and semantics of PSPARQL, let us briefly introduce the notion of regular expression patterns (cf. [1] for detailed discussion).

2.2.1 Regular Expressions

Regular expressions are patterns used to describe languages (i.e., sets of strings) from a given alphabet. Let $\Sigma = \{a_1, ..., a_n\}$ be an alphabet. A *string/word* is a finite sequence of symbols from the alphabet Σ. A *language* $\mathcal{L}$ is a set of words over Σ which is a subset of Σ^*, i.e, $\mathcal{L}(\Sigma) \subseteq \Sigma^*$. A word can be either empty ϵ or a sequence of alphabet symbols $a_1...a_n$. If $A = a_1...a_n$ and $B = b_1...b_m$ are two words over some alphabet Σ, then $A.B$ is a word over the same alphabet defined as: $A.B = a_1...a_n b_1...b_m$.

Definition 1 (Regular expression pattern)**.** *Given an alphabet Σ and a set of variables V, a regular expression $\mathcal{R}(\Sigma, V)$ can be constructed inductively as follows:*

$$e := uri \mid x \mid e_1 \shortmid e_2 \mid e_1.e_2 \mid e^+ \mid e^*$$

Where $e \in \mathcal{R}(\Sigma, V)$ and x denotes a variable, $e_1 \shortmid e_2$ denotes disjunction, $e_1.e_2$ denotes concatenation, e^+ denotes positive closure, and e^* denotes Kleene closure. Let U be a set of URIs and V a set of variables, a regular expression over $\mathcal{R}(U, V)$ can be used to define a language over the alphabet $U \cup V$.

The only difference between the syntax of SPARQL and PSPARQL is on triple patterns. In this study, we refer to both SPARQL and PSPARQL queries as SPARQL unless explicitly stated. Triple patterns in PSPARQL contain regular expressions in property positions instead of only URIs or variables as it is the case in SPARQL. Queries are formed based on the notion of query patterns defined inductively from triple patterns: a tuple $t \in \text{UBV} \times e \times \text{UBLV}$, with V a set of variables disjoint from UBL and e a regular expression pattern defined over U and V, is called a triple pattern. Triple patterns grouped together using connectives AND and UNION[2] form *graph patterns* (a.k.a query patterns). A set of triple patterns is called basic graph pattern. We use an abstract syntax that can be translated into μ-calculus.

Definition 2. *A SPARQL query pattern q is inductively defined as follows:*

$$\begin{aligned} q &= \ t \in \text{UBV} \times e \times \text{UBLV} \mid q_1 \ \textit{AND}\ q_2 \mid q_1 \ \textit{UNION}\ q_2 \\ e &= \ uri \mid x \mid e \shortmid e' \mid e \cdot e' \mid e^+ \mid e^* \end{aligned}$$

Definition 3. *A SPARQL SELECT query is a query of the form $q(\overrightarrow{w})$ where $\overrightarrow{w}$ is a tuple of variables in V which are called distinguished variables, and q is a query pattern.*

Example 3 (SPARQL queries)**.** *Consider the following queries $q(?x)$ and $q'(?x)$ on the graph of Example 1 and 2:*

```
PREFIX ex: <http://www.example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x WHERE {
            ?x  type Person .
 }
```

[2]We do not consider OPTIONAL and FILTER query patterns as containment over full SPARQL (equally expressive as relational algebra [2]) is undecidable.

and

```
PREFIX ex: <http://www.example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x WHERE {
       { ?x  ?p  ?y .
         ?p  rdfs:subPropertyOf*.rdfs:domain.rdfs:subClassOf* Person . }
         UNION
       { ?y  ?p  ?x .
         ?p  rdfs:subPropertyOf*.rdfs:range.rdfs:subClassOf* Person . }
 }
```

Definition 4 (SPARQL under RDFS entailment semantics)**.** *Given an RDF graph G and a basic graph pattern P, a partial mapping function ρ is a solution for G and P under RDFS-entailment, $\rho \in [\![P]\!]_G$, if:*

- *the domain of ρ is exactly the set of variable in P, i.e., $dom(\rho) = V(P)$,*
- *terms in the range of ρ occur in G,*
- *If P', obtained from P by replacing blank nodes with either URIs, blank nodes, or RDF literals is such that: the RDF graph $sk(\rho(P'))$ is RDFS-entailed by $sk(G)$. The function $sk(.)$ replaces blank nodes with fresh URIs (URIs that are neither in the queried graph nor in the query).*

Since SPARQL's entailment regimes only change the evaluation of basic graph patterns, the evaluation of query patterns can be defined in the standard way [21, 19]. The evaluation of query patterns over an RDF graph G is defined inductively:

$$[\![.]\!]_G : q \rightarrow 2^{\mathrm{V} \times \mathrm{UBL}}$$
$$[\![q_1 \text{ AND } q_2]\!]_G = [\![q_1]\!]_G \bowtie [\![q_2]\!]_G$$
$$[\![q_1 \text{ UNION } q_2]\!]_G = [\![q_1]\!]_G \cup [\![q_2]\!]_G \qquad [\![q(\overrightarrow{w})]\!]_G = \pi_{\overrightarrow{w}}([\![q]\!]_G)$$

The projection operator $\pi_{\overrightarrow{w}}$ selects only those part of the mappings relevant to variables in $\overrightarrow{w}$. For detailed discussions we refer the reader to [13, 1].

The semantics of PSPARQL queries is given by a partial mapping function $\rho : V \mapsto UBL$. The domain of ρ, $dom(\rho)$, is the subset of V on which ρ is defined. Two mappings ρ_1 and ρ_2 are said to be *compatible* if $\forall x \in dom(\rho_1) \cap dom(\rho_2)$, $\rho_1(x) = \rho_2(x)$. Hence, $\rho_1 \cup \rho_2$ is also a mapping. This allows for defining the join, union, and difference operations between two sets of mappings M_1, and M_2 as shown below:

$$M_1 \bowtie M_2 = \{\rho_1 \cup \rho_2 \mid \rho_1 \in M_1, \rho_2 \in M_2 \text{ are compatible mappings}\}$$
$$M_1 \cup M_2 = \{\rho \mid \rho \in M_1 \text{ or } \rho \in M_2\}$$
$$M_1 \setminus M_2 = \{\rho \in M_1 \mid \forall \rho_1 \in M_2, \rho \text{ and } \rho_1 \text{ are not compatible}\}$$

Now, we are ready to define the evaluation of PSPARQL triple patterns recursively as follows:

$$
\begin{aligned}
[\![(x, uri, y)]\!]_G &= \{\rho \mid (\rho(x), \rho(uri), \rho(y)) \in G\} \\
[\![(x, z, y)]\!]_G &= \{\rho \mid (\rho(x), \rho(z), \rho(y)) \in G\} \\
[\![(x, e \shortmid e', y)]\!]_G &= [\![(x, e, y)]\!]_G \cup [\![(x, e', y)]\!]_G \\
[\![(x, e \cdot e', y)]\!]_G &= [\![(x, e, n)]\!]_G \bowtie [\![(n, e', y)]\!]_G \\
[\![(x, e^+, y)]\!]_G &= \{\rho \mid \exists (n_0, e, n_1), (n_1, e, n_2), \ldots, (n_{k-1}, e, n_k) \in G \\
&\qquad \text{such that } n_0 = \rho(x), n_k = \rho(y) \text{ and } e \cdots e \in \mathcal{L}(e^+)\} \\
[\![(x, e^*, y)]\!]_G &= \{\rho \mid \rho(x) = \rho(y)\} \cup [\![(x, e^+, y)]\!]_G
\end{aligned}
$$

The evaluation of query patterns over an RDF graph G is inductively defined by:

$$
\begin{aligned}
[\![.]\!]_G &: q \rightarrow 2^{\mathrm{V} \times \mathrm{UBL}} \\
[\![q_1 \text{ AND } q_2]\!]_G &= [\![q_1]\!]_G \bowtie [\![q_2]\!]_G \\
[\![q_1 \text{ UNION } q_2]\!]_G &= [\![q_1]\!]_G \cup [\![q_2]\!]_G \\
[\![q\{\overrightarrow{w}\}]\!]_G &= \pi_{\overrightarrow{w}}([\![q]\!]_G)
\end{aligned}
$$

Where the projection operator $\pi_{\overrightarrow{w}}$ selects only those part of the mappings relevant to variables in $\overrightarrow{w}$.

Given an RDF graph G and a SPARQL query q, the answers of q under the RDFS-entailment regime are obtained through materialization or query rewriting. In the case of materialization, all the RDFS inferences are materialized. Hence, q can be evaluated over the materialized graph using subgraph matching. On the other hand, in the case of query rewriting, q is rewritten using the schema in the queried graph. Thus, the rewritten query can be evaluated over G using subgraph matching (simple RDF entailment).

Example 4. *The answers to query q and q' of Example 3 on graphs G of Example 1 and $cl(G)$ of Example 2 are: $[\![q]\!]_G = \emptyset$ but $[\![q]\!]_{cl(G)} = \{john, mary\}$. On the other hand, $[\![q']\!]_G = \{john, mary\}$. Therefore, $[\![q]\!]_{cl(G)} = [\![q']\!]_G$. Clearly, $[\![q]\!]_G \subseteq [\![q']\!]_G$.*

Example 5. *The answers to query q and q' (under simple entailment semantics) of Example 3 on graphs G of Example 1 and $cl(G)$ of Example 2 are: $[\![q]\!]_G = \emptyset$ but $[\![q]\!]_{cl(G)} = \{john, mary\}$ and $[\![q']\!]_G = \{john, mary\}$. Thus, $[\![q]\!]_{cl(G)} = [\![q']\!]_G$. Clearly, $[\![q]\!]_G \subseteq [\![q']\!]_G$. Note also that, q when evaluated over G under the RDFS entailment semantics is equivalent to q' evaluated under simple entailment semantics.*

Beyond these particular examples, the goal of query containment is to determine whether this holds for any graph.

Definition 5 (Containment)**.** *Given an RDFS schema $\mathcal{S}$ and queries q and q' with the same arity, q is contained in q' under the RDFS entailment regime, denoted $q \sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$, iff for any graph G satisfying the schema $\mathcal{S}$, $[\![q]\!]_G \subseteq [\![q']\!]_G$.*

Definition 6 (Equivalence)**.** *Two queries q and q' are equivalent, $q \equiv q'$, iff $q \sqsubseteq q'$ and $q' \sqsubseteq q$.*

Complexity: The evaluation of SPARQL queries (also under the RDFS entailment regime) is proved to be PSPACE-complete. However, the evaluation problem is NP-complete for the fragment containing only AND and UNION query patterns [19, 1, 13].

To determine containment, SPARQL queries are encoded as μ-calculus formulas, next we present a brief introductory about this logic.

2.3 μ-calculus

The modal μ-calculus [18] is an expressive logic which adds recursive features to modal logic using fixpoint operators. The syntax of the μ-calculus is composed of countable sets of *atomic propositions* AP, a set of *nominals* Nom, a set of *variables* Var, a set of *programs* Prog for navigating in graphs. A μ-calculus formula, φ, can be defined inductively as follows:

$$\varphi \quad ::= \quad \top \mid \bot \mid p \mid X \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X \varphi \mid \nu X \varphi$$

where $p \in AP, X \in Var$ and $a \in Prog$ is either an atomic program or its converse $\bar{a}$. The greatest and least fixpoint operators (ν and μ), respectively introduce general and finite recursion in graphs [18].

The semantics of the μ-calculus is given over a transition system, $K = (S, R, L)$ where S is a non-empty set of nodes, $R : Prog \to 2^{S \times S}$ is the transition function, and $L : AP \to 2^S$ assigns a set of nodes to each atomic proposition or nominal where it holds, such that $L(p)$ is a *singleton* for each nominal p. For converse programs, R can be extended as $R(\bar{a}) = \{(s', s) \mid (s, s') \in R(a)\}$. Besides, a valuation function $V : \text{Var} \to 2^S$ is used to assign a set of nodes to each variable. For a valuation V, variable X, and a set of nodes $S' \subseteq S$, $V[X/S']$ is the valuation that is obtained from V by assigning S' to X. The semantics of a formula, in terms of a transition system K (a.k.a. Kripke structure) and a valuation function, is represented by $[\![\varphi]\!]^K_V$. The semantics of basic μ-calculus formulae is defined as follows:

$$\begin{aligned}
[\![\top]\!]^K_V &= S \qquad [\![\bot]\!]^K_V = \emptyset \\
[\![p]\!]^K_V &= L(p), p \in AP \cup Nom,\ L(p) \text{ is singleton for } p \in Nom \\
[\![X]\!]^K_V &= V(X), X \in Var \qquad [\![\neg\varphi]\!]^K_V = S \backslash [\![\varphi]\!]^K_V \qquad [\![\top]\!]^K_V = S \\
[\![\varphi \wedge \psi]\!]^K_V &= [\![\varphi]\!]^K_V \cap [\![\psi]\!]^K_V, \quad [\![\varphi \vee \psi]\!]^K_V = [\![\varphi]\!]^K_V \cup [\![\psi]\!]^K_V \\
[\![\langle a \rangle \varphi]\!]^K_V &= \{s \in S \mid \exists s' \in S.(s, s') \in R(a) \ \wedge s' \in [\![\varphi]\!]^K_V\} \\
[\![[a]\varphi]\!]^K_V &= \{s \in S \mid \forall s' \in S.(s, s') \in R(a) \Rightarrow s' \in [\![\varphi]\!]^K_V\} \\
[\![\mu X \varphi]\!]^K_V &= \bigcap \{S' \subseteq S \mid [\![\varphi]\!]^K_{V[X/S']} \subseteq S'\} \\
[\![\nu X \varphi]\!]^K_V &= \bigcup \{S' \subseteq S \mid S' \subseteq [\![\varphi]\!]^K_{V[X/S']}\}
\end{aligned}$$

3 RDF Graphs as Transition Systems

μ-calculus formulas are interpreted over labeled transition systems. Thus, we propose an encoding of an RDF graph as a transition system in which nodes correspond to RDF entities and RDF triples. Edges relate entities to the triples they occur in. Different edges are used for distinguishing the functions (subject, object, predicate). Expressing predicates as nodes, instead of atomic programs, makes it possible to deal with full RDF expressiveness in which a predicate may also be the subject or object of a statement.

Definition 7 (Transition system associated to an RDF graph [10])**.** *Given an RDF graph, $G \subseteq UB \times U \times UBL$, the transition system associated to G, $\sigma(G) = (S, R, L)$ over $AP = \text{UBL} \cup \{s', s''\}$, is such that:*

- *$S = S' \cup S''$ with S' and S'' the smallest sets such that $\forall u \in U_G, \exists n^u \in S'$, $\forall b \in B_G, \exists n^b \in S'$, and $\forall t \in G, \exists n^t \in S''$,*
- *$\forall t = (s, p, o) \in G$, $\langle n^s, n^t \rangle \in R(s)$, $\langle n^t, n^p \rangle \in R(p)$, and $\langle n^t, n^o \rangle \in R(o)$,*

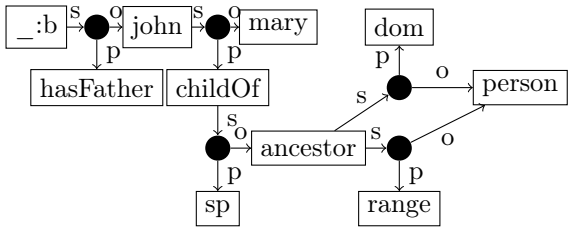


Figure 1: Transition system encoding the RDF graph of Example 1. Nodes in S'' are black anonymous nodes; nodes in S' are the other nodes (d-transitions are not displayed).

- $L : \text{AP} \to 2^S$; $\forall u \in U_G, L(u) = \{n^u\}$, $\forall b \in B_G, L(b) = S'$, $L(s') = S'$, $\forall l \in L_G, L(l) = \{n^l\}$ *and* $L(s'') = S''$,
- $\forall n^t, n^{t'} \in S''$, $\langle n^t, n^{t'} \rangle \in R(d)$.

The program d is introduced to render each triple accessible to the others and thus facilitate the encoding of queries. The function σ associates what we call a *restricted transition system* to any RDF graph. Formally, we say that a transition system K is a *restricted transition system* iff there exists an RDF graph G such that $K = \sigma(G)$.

A restricted transition system is thus a bipartite graph composed of two sets of nodes: S', those corresponding to RDF entities, and S'', those corresponding to RDF triples. For example, Figure 1 shows the restricted transition system associated with the graph of Example 1. When checking for query containment, we consider the following restrictions:

- The set of programs is fixed: $Prog = \{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$.
- A model must be a restricted transition system.

The latter constraint can be expressed in the μ-calculus as follows:

Proposition 1 (RDF restriction on transition systems [10])**.** *A formula φ is satisfied by some restricted transition system if and only if $\varphi \wedge \varphi_r$ is satisfiable by some transition system, i.e. $\exists K_r [\![\varphi]\!]^{K_r} \neq \emptyset \iff \exists K [\![\varphi \wedge \varphi_r]\!]^{K} \neq \emptyset$, where:*

$$\varphi_r = \quad \nu X.\theta \wedge \kappa \ \wedge (\neg\langle d\rangle\top \vee \langle d\rangle X)$$

in which $\theta = \langle\bar{s}\rangle s' \wedge \langle p\rangle s' \wedge \langle o\rangle s' \wedge \neg\langle s\rangle\top \wedge \neg\langle\bar{p}\rangle\top \wedge \neg\langle\bar{o}\rangle\top$ and $\kappa = [\bar{s}]\xi \wedge [p]\xi \wedge [o]\xi$ with

$$\xi = (\neg\langle\bar{s}\rangle\top \wedge \neg\langle o\rangle\top \wedge \neg\langle p\rangle\top \wedge \neg\langle d\rangle\top \wedge \neg\langle\bar{d}\rangle\top \wedge \neg\langle s\rangle s' \wedge \neg\langle\bar{o}\rangle s' \wedge \neg\langle\bar{p}\rangle s').$$

The formula φ_r ensures that θ and κ hold in every node reachable by a d edge, i.e. in every s'' node. The formula θ forces each s'' node to have a subject, predicate and object. The formula κ navigates from a s'' node to every reachable s' node, and forces the latter not to be directly connected to other subject, predicate or object nodes.

If a μ-calculus formula ψ appears under the scope of a least μ or greatest ν fixed point operator over all the programs $\{s, p, o, d, \bar{s}, \bar{p}, \bar{o}, \bar{d}\}$ as, $\mu X.\psi \vee \langle s\rangle X \vee \langle p\rangle X \vee \cdots$ or $\nu X.\psi \wedge \langle s\rangle X \wedge \langle p\rangle X \wedge \cdots$ hen, for the sake of legibility, we denote the recursion components of the respective formulae as $murec(X)$ for the μ recursion part and $nurec(X)$ for the ν recursion part.

4 Encoding SPARQL Queries

In this section, we show how to encode queries as μ-calculus formulas. Then, in the next section, we use this encoding to test query containment under the RDFS entailment regime. Before discussing the encoding procedure, we briefly assess the issue of blank nodes. Blank nodes are existential variables that denote the existence of unnamed resources. Their definition matches the definition of non-distinguished variables in a query. Thus, blank nodes in the queries can be considered as non-distinguished variables. As a result, every occurrence of a blank node in the query is replaced by a fresh variable.

Queries are translated into μ-calculus formulas. The principle of the translation is that each triple pattern is associated with a sub-formula stating the existence of the triple somewhere in the graph. Hence, they are quantified by μ (least fixed point) so as to put them out of the context of a state. In this translation, variables are replaced by nominals which will be satisfied when they are at the corresponding position in such triple relations. A function called $\mathcal{A}$ is used to encode queries inductively on the structure of query patterns. AND and UNION are translated into boolean connectives $\wedge$ and $\vee$, respectively. When encoding $q \sqsubseteq q'$, we call q left-hand side query and q' right-hand side query. Cyclic dependencies among the non-distinguished variables in the query on the right-hand side create problems in the encoding process: because variables in cycles cannot be simply encoded using atomic propositions (APs) or $\top$. As APs can be true in several nodes in the transition system (resulting in the loss of connectedness). Thus, we provide separate encodings for q and q'.

Encoding left-hand side query: to encode the left-hand side query, one proceeds by encoding the distinguished or non-distinguished variables and constants using nominals. Basically, the variables and constants are frozen (i.e., equivalent to obtaining a canonical instance of the query). Afterwards, a recursive function $\mathcal{A}$ is used to inductively construct a formula. Regular expression patterns that appear in the query are encoded using the function $\mathcal{R}$. It takes two arguments (the predicate which is a regular expression pattern and the object of a triple).

$$\mathcal{A}\big((x,e,z)\big) = \mu X.\big(\langle \bar{s}\rangle x \wedge \mathcal{R}(e,z)\big) \vee murec(X)$$

$$\mathcal{A}(q_1 \text{ AND } q_2) = \mathcal{A}(q_1) \wedge \mathcal{A}(q_2) \qquad \mathcal{A}(q_1 \text{ UNION } q_2) = \mathcal{A}(q_1) \vee \mathcal{A}(q_2)$$

$$\mathcal{R}(uri, y) = \langle p\rangle uri \ \wedge \langle o\rangle y \qquad \mathcal{R}(x,y) = \langle p\rangle x \wedge \langle o\rangle y$$

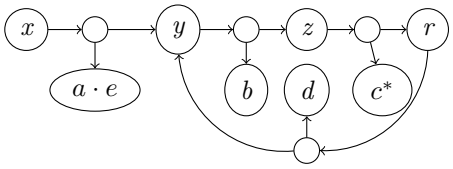
$$\mathcal{R}(e \shortmid e', y) = (\mathcal{R}(e,y) \vee \mathcal{R}(e',y)) \qquad \mathcal{R}(e \cdot e', y) = \mathcal{R}(e, \langle s\rangle \mathcal{R}(e',y))$$

$$\mathcal{R}(e^+, y) = \mu X.\mathcal{R}(e,y) \vee \mathcal{R}(e, \langle s\rangle X) \qquad \mathcal{R}(e^*, y) = \mathcal{R}(e^+, y) \vee \langle \bar{s}\rangle y$$

In order to encode the right-hand side query, we need the notion of cyclic queries.

Definition 8 (Cyclic Query)**.** *A SPARQL query is referred to as cyclic iff a transition graph induced from the query patterns is cyclic. The transition graph*[3] *is constructed in the same way as done in Definition 7.*

Example 6. *Consider* $q(x) = (x, a \cdot e, y)$ AND (y, b, z) AND (z, c^*, r) AND (r, d, y) *which is cyclic, as shown graphically,*



[3]The transition graph is similar to the tuple-graph used in [5] to detect dependency among variables.

Encoding right-hand side query: the encoding of the right-hand side query q is different from that of the left due to the non-distinguished variables that appear in cycles in the query. The distinguished variables and constants are encoded as nominals whereas the non-distinguished variables are encoded as:

- if a non-distinguished variable appears only once, then it is encoded as $\top$.
- if a non-distinguished variable appears multiple times, then one performs the subsequent steps:
  1. for each $t_i \in q$, $t(t_i) = n_i$, i.e., introduce a nominal for each triple,
  2. for each $z \in t_i = (x_i, e_i, y_i) \in q$, a set of mappings containing formula assignments are generated as:

$$m_i = \{z \mapsto \psi \mid \begin{cases} \psi = \varphi(s, e_i) \text{ if } subject(z) \wedge e_i \notin var(q) \\ \psi = \langle s \rangle t(t_i) \text{ if } subject(z) \wedge e_i \in var(q) \\ \psi = \varphi(o, e_i) \text{ if } object(z) \wedge e_i \notin var(q) \\ \psi = \langle \bar{o} \rangle t(t_i) \text{ if } object(z) \wedge e_i \in var(q) \\ \psi = \langle \bar{p} \rangle t(t_i) \text{ if } predicate(z) \wedge e_i \in var(q) \end{cases} \}$$

  s and o denote subject and object of a triple and φ is defined as:

$$\begin{aligned} \varphi(s, a) &= \langle s \rangle \langle p \rangle a & \varphi(o, a) &= \langle \bar{o} \rangle \langle p \rangle a \\ \varphi(s, a \cdot b) &= \varphi(s, a) & \varphi(o, a.b) &= \varphi(o, b) \\ \varphi(s, a \intercal b) &= \big(\varphi(s, a) \vee \varphi(s, b)\big) & \varphi(o, a \intercal b) &= \big(\varphi(o, a) \vee \varphi(o, b)\big) \\ \varphi(s, a^+) &= \varphi(s, a) & \varphi(o, a^+) &= \varphi(o, a) \\ \varphi(s, a^*) &= \varphi(s, a) & \varphi(o, a^*) &= \varphi(o, a) \end{aligned}$$

  Note that there is an exponential number of m_i's in terms of the number of non-distinguished variables. More precisely, there are at most $\mathcal{O}(k^n)$ mappings, where n is the number of triples in which non-distinguished variables appear and k is the number of non-distinguished variables.

- finally function $\mathcal{A}$ works inductively on the query structure using m to generate the formula. As for the left-hand side query, $\mathcal{R}$ is used to produce the encodings of regular expressions.

$$\mathcal{A}(q, m) = \bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i) \qquad d(m, x) = \begin{cases} \psi & \text{if } (x \mapsto \psi) \in m \\ \top & \text{if } unique(x) \\ x & \text{otherwise} \end{cases}$$

$$\begin{aligned} \mathcal{A}\big((x, e, z), m\big) &= \mu X.\big(\langle \bar{s} \rangle d(m, x) \wedge \mathcal{R}(d(m, e), d(m, e))\big) \vee murec(X) \\ \mathcal{A}(q_1 \text{ AND } q_2, m) &= \mathcal{A}(q_1, m) \wedge \mathcal{A}(q_2, m) \\ \mathcal{A}(q_1 \text{ UNION } q_2, m) &= \mathcal{A}(q_1, m) \vee \mathcal{A}(q_2, m) \end{aligned}$$

Example 7 (Encoding queries)**.** *Consider the encoding of $q \sqsubseteq q'$, where*

$$q(x, z) = (x, (c \intercal d) \cdot (a \intercal b), z) \qquad q'(x, z) = (x, (c \intercal d), y) \text{ AND } (y, a \intercal b, z)$$

- *The encoding of q is obtained by freezing the query and recursively constructing the formula using $\mathcal{A}$.*

$$\begin{aligned}\mathcal{A}(q) &= \mu X.\langle\bar{s}\rangle x \wedge \mathcal{R}\big((c \shortmid d)\cdot(a \shortmid b), z\big) \vee murec(X)\\ &= \mu X.\langle\bar{s}\rangle x \wedge \mathcal{R}\big((c \shortmid d), \langle s\rangle\mathcal{R}(a \shortmid b, z)\big) \vee murec(X)\\ &= \mu X.\langle\bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle\langle s\rangle\mathcal{R}(a \shortmid b, z) \vee murec(X)\\ &= \mu X.\langle\bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle\langle s\rangle((\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z) \vee murec(X)\end{aligned}$$

- *The encoding of q' is as follows:*
  - *the constants and distinguished variables are encoded as nominals,*
  - *$y \in var(q')$ is encoded as $\varphi(o,(c \shortmid d))$, since y is an object of the triple $(x,(c \shortmid d),y)$. Hence, $m_1 = \{y \mapsto (\langle\bar{o}\rangle\langle p\rangle c \vee \langle\bar{o}\rangle\langle p\rangle d)\}$. On the other hand, y can also be encoded as $\varphi(s,(a \shortmid b))$, since y is a subject of the triple $(y, a \shortmid b, z)$. Thus, we get $m_2 = \{y \mapsto (\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b)\}$.*
  - *finally, we use $\mathcal{A}$ to encode q' recursively, $\mathcal{A}(q', m)$*

$$\begin{aligned}&= \mathcal{A}(q', m_1) \vee \mathcal{A}(q', m_2)\\ &= \big(\mu X.\langle\bar{s}\rangle d(m_1, x) \wedge \mathcal{R}\big((c \shortmid d), d(m_1, y)\big) \vee murec(X)\\ &\qquad \wedge \mu Y.\langle\bar{s}\rangle d(m_1, y) \wedge \mathcal{R}\big((a \shortmid b), d(m_1, z)\big) \vee murec(Y)\big)\vee\\ &\quad \big(\mu X.\langle\bar{s}\rangle d(m_2, x) \wedge \mathcal{R}\big((c \shortmid d), d(m_2, y)\big) \vee murec(X)\\ &\qquad \wedge \mu Y.\langle\bar{s}\rangle d(m_2, y) \wedge \mathcal{R}\big((a \shortmid b), d(m_2, z)\big) \vee murec(Y)\big)\\ &= \big(\mu X.\langle\bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle(\langle\bar{o}\rangle\langle p\rangle c \vee \langle\bar{o}\rangle\langle p\rangle d) \vee murec(X)\\ &\qquad \wedge \mu Y.\langle\bar{s}\rangle(\langle\bar{o}\rangle\langle p\rangle c \vee \langle\bar{o}\rangle\langle p\rangle d) \wedge (\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z \vee murec(Y)\big)\vee\\ &\quad \big(\mu X.\langle\bar{s}\rangle x \wedge (\langle p\rangle c \vee \langle p\rangle d) \wedge \langle o\rangle(\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b) \vee murec(X)\\ &\qquad \wedge \mu Y.\langle\bar{s}\rangle(\langle s\rangle\langle p\rangle a \vee \langle s\rangle\langle p\rangle b) \wedge (\langle p\rangle a \vee \langle p\rangle b) \wedge \langle o\rangle z \vee murec(Y)\big)\end{aligned}$$

Example 8 (Containment test)**.** *We show containment of the following queries: select all descendants and ancestors (q) whose names are "john" and (q') who share the same name.*
$q(x,y) = (x, name, \text{“}john\text{”})$ AND $(x, ancestor^*, z)$ AND $(z, name, \text{“}john\text{”})$
$q'(x,y) = (x, name, y)$ AND $(x, ancestor^*, z)$ AND $(z, name, y)$
We proceed by first obtaining their encodings. Consider the encoding of $q \sqsubseteq q'$, we encode triple patterns using θ and $m = \{y \mapsto \langle\bar{o}\rangle name\}$.

$$\begin{aligned}\mathcal{A}(q) &= \big(\mu X.\theta(x, name, \text{“}john\text{”}) \vee murec(X)\big)\wedge\\ &\qquad \big(\mu X.\theta(x, ancestor^*, z) \vee murec(X)\big)\wedge\\ &\qquad \big(\mu X.\theta(z, name, \text{“}john\text{”}) \vee murec(X)\big)\\ \neg\mathcal{A}(q', m) &= \big(\nu X.\neg\theta(x, name, \langle\bar{o}\rangle name) \wedge nurec(X)\big)\vee\\ &\qquad \big(\nu X.\neg\theta(x, ancestor^*, z) \wedge nurec(X)\big)\vee\\ &\qquad \big(\nu X.\neg\theta(z, name, \langle\bar{o}\rangle name) \wedge nurec(X)\big)\end{aligned}$$

The formula $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)$ is unsatisfiable because $\mathcal{A}(q)$ demands its model to satisfy the encoding of each triple pattern somewhere in the transition system. On the contrary, the formula $\neg\mathcal{A}(q', m)$ requests this model to satisfy the negation of the encoding of the triples in the entire transition system. Hence, this leads to a contradiction and no such model exists for the formula. Therefore, $q \sqsubseteq q'$. On the other hand, it can be verified similarly to arrive at $q' \not\sqsubseteq q$.

5 Query Containment under RDFS Entailment

In the following, we propose three approaches to determine query containment under the RDFS entailment regime: encoding the RDFS semantics, query rewriting, and encoding the schema.

5.1 Encoding the RDFS Semantics

When queries are evaluated under the RDFS entailment regime, the queried graph is materialized or saturated using RDFS inference rules (or simply rules) and the schema. Henceforth, implicit or inferred triples are considered when computing the result of the query. Since no specific graphs are considered when dealing with containment, we encode schema and rules. In addition, blank nodes that appear in the schema graph are skolemized, i.e., replaced by fresh constants that do not appear neither in the queries nor schema.

Definition 9. *The encoding of an RDF schema graph $S = \{t_1, \cdots, t_n\}$ is produced by encoding each schema triple $t_i = (x, y, z) \in S$ such that:*

$$\Phi_S = \bigwedge_{i=1 \wedge t_i \in S}^{n} \big(\mu X.(\langle \bar{s} \rangle x \wedge \langle p \rangle y \wedge \langle o \rangle z) \vee murec(X)\big)$$

where x, y, and z are atomic propositions corresponding to triple elements.

Definition 10 (Encoding inference rules)**.** *The μ-calculus encoding of RDFS inference rules of (1)–(9) is the disjunction of formulas (1) to (6) such that:*

$$\begin{aligned}
(1)\quad & \nu X.\big(\theta(x, \texttt{sc}, \theta(y, \texttt{sc}, z)) \Rightarrow \theta(x, \texttt{sc}, z)\big) \wedge nurec(X)\\
(2)\quad & \nu X.\big(\theta(x, \texttt{type}, \theta(a, \texttt{sc}, b)) \Rightarrow \theta(x, \texttt{type}, b)\big) \wedge nurec(X)\\
(3)\quad & \nu X.\big(\theta(x, \texttt{sp}, \theta(y, \texttt{sp}, z)) \Rightarrow \theta(x, \texttt{sp}, z)\big) \wedge nurec(X)\\
(4)\quad & \nu X.\big(\theta(x, \theta(a, \texttt{sp}, b), y) \Rightarrow \theta(x, b, y)\big) \wedge nurec(X)\\
(5)\quad & \nu X.\big(\theta(x, \theta(a, \texttt{dom}, b), y) \Rightarrow \theta(x, \texttt{type}, b)\big) \wedge nurex(X)\\
(6)\quad & \nu X.\big(\theta'(x, \theta(a, \texttt{range}, b), y) \Rightarrow \theta(y, \texttt{type}, b)\big) \wedge nurex(X)
\end{aligned}$$

$$\theta(x, y, z) = x \wedge \langle s \rangle(\langle p \rangle y \wedge \langle o \rangle z) \qquad \theta'(x, y, z) = z \wedge \langle \bar{o} \rangle\big(\langle p \rangle(y \wedge \langle \bar{s} \rangle x)$$

We denote this formula by Φ_R.

So far, we have produced the encoding of SPARQL queries $\mathcal{A}(q)$ and $\mathcal{A}(q, m)$, RDFS inference rules Φ_R, and schema triples (axioms) Φ_S. In the following, we reduce query containment to unsatisfiability in μ-calculus and prove the correctness of this reduction.

Theorem 1. *Given a query $q(\overrightarrow{w})$, there exists an RDF graph G such that $[\![q(\overrightarrow{w})]\!]_G \neq \emptyset$.*

Proof. (*Sketch*) From any query it is possible to build an homomorphic graph by collecting all triples connected by AND and only those at the left of UNION (replacing variables by blanks). This graph is consistent as all RDF graphs [14]. It is thus a graph satisfying the query. □

Lemma 1. *Given an RDFS graph S and a graph G,*

$$G \models S \Leftrightarrow [\![\Phi_R \wedge \Phi_S]\!]^{\sigma(G)} \neq \emptyset$$

Proof. ($\Rightarrow$) $G \models S$ implies $S \subseteq G$. Also, $S \subseteq cl(G)$, where $cl(G)$ is the closure of G obtained by applying exhaustively the inference rules on the schema S and graph G. Without of loss generality, we can assume that $cl(G) \subseteq G$. Clearly it follows that, $[\![\Phi_S]\!]^{\sigma(G)} \neq \emptyset$. Since the rules are exhausted to compute $cl(G)$ and this graph is contained in G, we have that Φ_R is true in $\sigma(G)$. Thus, it is apparent that $[\![\Phi_R \wedge \Phi_S]\!]^{\sigma(G)} \neq \emptyset$.
($\Leftarrow$) From $[\![\Phi_R \wedge \Phi_S]\!]^{\sigma(G)} \neq \emptyset$, it follows that $[\![\Phi_R]\!]^{\sigma(G)} \neq \emptyset$ and $[\![\Phi_S]\!]^{\sigma(G)} \neq \emptyset$. The later dictates that there exists an encoding of an RDFS graph $\sigma(S)$ making up the schema as a part of the transition system $\sigma(G)$. Consequently, $S \subseteq G$ and thus $G \models S$. On the other hand, applying the inference rules to G and S results in $cl(G) \subseteq G$. Therefore, $G \models S$. □

Lemma 2. *For any SPARQL query q, q is satisfiable iff $\mathcal{A}(q)$ and $\mathcal{A}(q, m)$ are satisfiable.*

Proof. If we prove for $\mathcal{A}(q, m)$, the proof for $\mathcal{A}(q)$ is immediate.
($\Rightarrow$) q is satisfiable implies there exists a graph G built from the canonical instance of q using a function f that satisfies it.

- if $(x, y, z) \in q$, then $f((x, y, z)) = (x, y, z) \in G$,
- if $(x, e, z) \in q$, then $f((x, e, z)) = (x, e, z) \in G$,
- if $(x, e \cdot e', z) \in q$, then $f((x, e, y)) \in G$ and $f((y, e', z)) \in G$,
- if $(x, e \mid e', z) \in q$, then $f((x, e, z)) \in G$ or $f((x, e', z)) \in G$,
- if $(x, e^+, z) \in q$, then $f((x, e, y_1)) \in G$ and ... and $f((y_n, e, z)) \in G$,
- if $(x, e^*, z) \in q$, then $f((x, e^+, z)) \in G$

Since G is an instance of q, G is a model of q (cf. Theorem 1). Now, we construct a transition system $\sigma(G) = (S, R, L)$ in the same way as is done in Definition 7. To prove that $\sigma(G)$ is a model of $\mathcal{A}(q, m)$, we consider two cases:

(i) when q is cyclic, and

(ii) when q is cycle-free

First, (i) consider when q is cyclic, in this case, its encoding is $\bigvee_{i=1}^{|m|} \mathcal{A}(q, m_i)$. From this encoding it is clear that nominals are introduced and are set to be true in S'' nodes. With these nominals, one can successfully create a formula that can encode multiply occurring non-distinguished variables. Henceforth, creating a formula that is satisfiable in cyclic models. Further, the disjuncts in the encoding guarantee that possible set of substitutions m capture the intended semantics of a cyclic query. One can verify that $\sigma(G)$ is a model of the disjuncts $\mathcal{A}(q, m_i)$, this is because nominals encoding the constants and distinguished variables are true in $\sigma(G)$ as they exist in G. Whereas the subformula obtained through φ guarantees that a nominal which is true in the triple node S'' and any labelling matches the node denoting the non-distinguished variable. Therefore, $\mathcal{A}(q, m)$ is satisfiable in $\sigma(G)$. To elaborate, if $l \in (x, y, z) \in q$

- for l either a distinguished variable or constant, l is satisfiable in $\sigma(G)$ since $[\![l]\!]^{\sigma(G)} \in L(l)$,
- for l a uniquely appearing non-distinguished variable, l is true in $\sigma(G)$ since $\top$ is true everywhere in the transition system,

- a multiply occurring non-distinguished variable l is true in $\sigma(G)$ since $\exists t \in S''.t \in L(n) \wedge t \in [\![n \wedge \langle o \rangle \top]\!]^{\sigma(G)}$. Where n is a nominal denoting the reified triple (x, y, z).
- for $l = e$ a regular expression, it is clear from the encoding that $\mathcal{R}(e, z)$ is satisfiable in $\sigma(G)$.

If (ii) q is cycle-free, then encoding the non-distinguished variables with $\top$ suffices to justify that $\sigma(G)$ is a model of its encoding.
($\Leftarrow$) Assume that $\mathcal{A}(q)$ is satisfiable. This implies there exists a transition system $K = (S, R, L)$ such that $[\![\mathcal{A}(q)]\!]^K \neq \emptyset$. We now create an RDF graph G from K as follows:

- if $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i) \wedge s_3 \in L(z_i)$, then $(x_i, y_i, z_i) \in G$. This case holds if x_i, y_i and z_i are either distinguished variables or constants. Note here that if x_i or y_i or z_i appear in another triple $t_j = (x_j, y_j, z_j) \in q$, then the equivalent item in t_j is replaced with the value of the corresponding entry in t_i.
- if $\forall s_1, s_2, s_3 \in S' \wedge t \in S''.(s_1, t) \in R(s) \wedge (t, s_2) \in R(p) \wedge (t, s_3) \in R(o)$ and for each triple $t_i = (x_i, y_i, z_i) \in q$ if $s_1 \in L(x_i) \wedge s_2 \in L(y_i)$, then $(x_i, y_i, c_i) \in G$ where c_i is a fresh constant. This case holds if z_i is a non-distinguished variable. Similarly, the case when x_i or y_i or both are variables can be worked out.

Since G is a technical construction obtained from q, it is holds that $[\![q]\!]_G \neq \emptyset$. Thus, q is satisfiable. □

For the sake of legibility, we denote $\Phi_R \wedge \Phi_S \wedge A(q) \wedge \neg A(q', m) \wedge \varphi_r$ by $\Phi(S, q, q')$.

Theorem 2 (Soundness and Completeness)**.** *Given SPARQL queries q and q' and a schema S, $\Phi(S, q, q')$ is unsatisfiable if and only if $q \sqsubseteq^S_{\text{rdfs}} q'$.*

Proof. ($\Rightarrow$) we prove the contrapositive, $q \not\sqsubseteq^S_{\text{rdfs}} q' \Rightarrow \Phi(S, q, q')$ is satisfiable. Assume there exists a graph G that entails the schema graph S, also assume that there exists a tuple $\overrightarrow{a} \in [\![q]\!]_G$ and $\overrightarrow{a} \notin [\![q']\!]_G$. We construct a restricted transition system K from G. Using Lemma 1, it is obvious that Φ_S is satisfiable in K. Besides, $[\![\varphi_r]\!]^K \neq \emptyset$ (cf. Proposition 1). Now let us use $\overrightarrow{a}$ to instantiate the distinguished variables in q and q'. Using the encodings of the instantiated queries and from Lemma 2, one deduces that $[\![\mathcal{A}(q)]\!]^K \neq \emptyset$ and $[\![\mathcal{A}(q', m)]\!]^K = \emptyset$. The later is not satisfiable in K because the nominals corresponding to the constants are not satisfied. Consequently, $[\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset$ and $\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)$ is satisfiable. Therefore, we arrive at $\Phi(S, q, q')$ is satisfiable.

($\Leftarrow$) we show that if $\Phi(S, q, q')$ is satisfiable, then $q \not\sqsubseteq_{rdfs} q'$. Consider a restricted transition system model K for $\Phi(S, q, q')$. We construct an RDF graph G from K. From Lemma 1, it follows that $G \models S$. Thus, it remains to verify that $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. To do so, we start from the assumption, $[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q', m)]\!]^K \neq \emptyset$. Subsequently, $[\![\mathcal{A}(q)]\!] \neq \emptyset$ and $[\![\mathcal{A}(q', m)]\!]^K = \emptyset$ because G contains all those triples that satisfy q and not q'. Besides, if q' contains a cycle, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied due to the ability, in a μ-calculus extended with nominals and converse, to express a formula that is satisfied in cyclic models. Therefore, $q \not\sqsubseteq^S_{\text{rdfs}} q'$. □

5.2 Query Rewriting

SPARQL query containment under RDFS entailment regime can be determined by rewriting queries using the RDFS inference rules (shown in equation (1)–(9)) and then reducing the encoding of the rewriting to unsatisfiability test. The rewriting is done using PSPARQL as explained in the following definition.

Definition 11 (SPARQL to PSPARQL)**.** *Given a SPARQL query q, a rewriting function τ produces its PSPARQL equivalent as follows:*

$$\begin{aligned}
\tau((s, sc, o)) &= (s, sc^+, o) \quad \tau((s, sp, o)) = (s, sp^+, o) \\
\tau((s, p, o)) &= (s, x, o) \ \textsc{and}\ (x, sp^*, p) \ \textit{such that}\ p \notin \{sc, sp, type\} \\
\tau((s, type, o)) &= (s, type.sc^*, o) \ \textsc{union}\ (s, x, y) \ \textsc{and}\ (x, \mathtt{sp}^*.\mathtt{dom}.\mathtt{sc}^*, o) \\
&\quad \textsc{union}\ (y, x, s) \ \textsc{and}\ (x, \mathtt{sp}^*.\mathtt{range}.\mathtt{sc}^*, o) \\
\tau((s, x, o)) &= (s, x, o) \ \textit{when } x \textit{ is a variable} \\
\tau(q_1 \ \textsc{and}\ q_2) &= \tau(q_1) \ \textsc{and}\ \tau(q_2) \quad \tau(q_1 \ \textsc{union}\ q_2) = \tau(q_1) \ \textsc{union}\ \tau(q_2)
\end{aligned}$$

Definition 12 (Containment under RDFS entailment)**.** *Given an RDF schema S, queries q and q', and a rewriting function τ. q is contained in q' under RDFS entailment, denoted $q \sqsubseteq^S_{\text{rdfs}} q'$, if and only if $\tau(q) \sqsubseteq^S \tau(q')$.*

Theorem 3 (Soundness and Completeness)**.** *Given an RDF schema $\mathcal{S}$ and SPARQL queries q and q', $q \sqsubseteq^S_{\text{rdfs}} q' \Leftrightarrow \Phi_S \wedge \mathcal{A}(\tau(q)) \wedge \neg\mathcal{A}(\tau(q'), m) \wedge \varphi_r$ is unsatisfiable.*

Proof. The proof of this theorem follows from that of Theorem 2. □

5.3 Encoding the Schema

In this approach, in order to determine query containment under the RDFS entailment regime, we encode the schema triples (axioms) as formulae. As a consequence, the encoding of the axioms constrains a model satisfying the formula. We consider *subclass*, *subproperty*, *domain*, *range*, and *transitivity* (Tr(sc) or Tr(sp)) schema axioms.

Definition 13. *Given a set of axioms $s_1, s_2, ..., s_n$ of a schema $\mathcal{S}$, the μ-calculus encoding of $\mathcal{S}$ is: $\eta(\mathcal{S}) = \eta(s_1) \wedge \eta(s_2) \wedge ... \wedge \eta(s_n)$.*
We use a function η to translate each s_i into an equivalent μ-calculus formula:

$$\begin{aligned}
\eta((C_1, \mathtt{sc}, C_2)) &= \nu X.\ (C_1 \Rightarrow C_2) \wedge nurec(X) \\
\eta((R_1, \mathtt{sp}, R_2)) &= \nu X.\ (R_1 \Rightarrow R_2) \wedge nurec(X) \\
\eta((R, \mathtt{dom}, C)) &= \nu X.\ \big(\langle s\rangle(\langle p\rangle R \Rightarrow \langle p\rangle\mathtt{type} \wedge \langle o\rangle C)\big) \wedge nurec(X) \\
\eta((R, \mathtt{range}, C)) &= \nu X.\ \big(\langle \bar{o}\rangle\langle p\rangle R \Rightarrow \langle s\rangle(\langle p\rangle\mathtt{type} \wedge \langle o\rangle C)\big) \wedge nurec(X) \\
\eta(Tr(\mathtt{sc})) &= \nu X.\big(\theta(x, \mathtt{sc}, \theta(y, \mathtt{sc}, z)) \Rightarrow \theta(x, \mathtt{sc}, z)\big) \wedge nurec(X) \\
\eta(Tr(\mathtt{sp})) &= \nu X.\big(\theta(x, \mathtt{sp}, \theta(y, \mathtt{sp}, z)) \Rightarrow \theta(x, \mathtt{sp}, z)\big) \wedge nurec(X)
\end{aligned}$$

Lemma 3. *Given a set of RDF schema axioms $\mathcal{C} = \{c_1, \cdots, c_n\}$, $\mathcal{C}$ has a model iff $\eta(\mathcal{C})$ is satisfiable.*

Proof. ($\Rightarrow$) assume that there exists a model $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ of $\mathcal{C}$ such that $\mathcal{I} \models \mathcal{C}$. We build a transition system $K = (S, R, L)$ from $\mathcal{I}$ using the following:

- for each element of the domain $e \in \Delta^{\mathcal{I}}$, we create a node $n^e \in S'$,
- for each atomic concept A, if $a \in A^{\mathcal{I}}$, then $(n^a, t) \in R(s), (t, n^{type}) \in R(p)$, $(t, n^A) \in R(o)$, $L(type), = n^{type}, L(A) = n^A$ and $L(a) = n^a$ where $t \in S''$,
- for each atomic role R, if $(x, y) \in R^{\mathcal{I}}$, then $\exists n^x, n^y, t, n^R$ such that $(n^x, t) \in R(s)$, $(t, n^R) \in R(p)$, and $(t, n^y) \in R(o)$ where $n^x, n^y, n^R \in S'$, $t \in S''$, and $L(x) = n^x$, $L(R) = n^R$, $L(y) = n^y$,

- $S = S' \cup S''$

To show that $\eta(\mathcal{C})$ is satisfiable in K. We proceed inductively on the construction of the formula. Since the axioms $c_1, \cdots, c_n$ are made of role or concept inclusions, we consider the following two cases:

- when $\eta(c_i) = \nu X.\big(\omega(C_1) \Rightarrow \omega(C_2)\big) \wedge nurec(X)$. Since $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, we get that $[\![\omega(C_1)]\!]^K \subseteq [\![\omega(C_2)]\!]^K$. And hence, $\omega(C_1) \Rightarrow \omega(C_2)$ is satisfiable in K. Besides, the general recursion ν guarantees that the constraint is satisfied in each state of the transition system. Therefore, $\eta(c_i)$ is satisfiable.
- when $\eta(c_i) = \nu X.\big(\omega(r_1) \Rightarrow \omega(r_2)\big) \wedge nurec(X)$. From $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$ we have that $\exists n^{r_1} \in L(r_1)$ implies $\exists n^{r_2} \in L(r_2)$ in K. Thus, $\exists s \in [\![\omega(r_1) \Rightarrow \omega(r_2)]\!]^K$. As K is a construction of $\mathcal{I}$, $\eta(c_i)$ is satisfiable in K.

Since K is a model of each $\eta(c_i)$, then $\eta(\mathcal{C})$ is satisfiable.
($\Leftarrow$) consider a transition system model K for $\eta(\mathcal{C})$. From K, we construct an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ and show that it is a model of $\mathcal{C}$.

- $\Delta^{\mathcal{I}} = S$, $A^{\mathcal{I}} = [\![A]\!]^K$ for each atomic concept A,
- $\top^{\mathcal{I}} = [\![\top]\!]^K$,
- $r^{\mathcal{I}} = \{(s, s') \mid \forall t \in [\![r]\!]^K \wedge t' \in S \wedge (s, t') \in R(s) \wedge (t', t) \in R(p) \wedge (t', s') \in R(o)\}$ for each atomic role r,

Consequently, formulas such as $\nu X.\big(\omega(r_1) \Rightarrow \omega(r_2)\big) \wedge nurec(X)$ and $\nu X.\big(\omega(C_1) \Rightarrow \omega(C_2)\big) \wedge nurec(X)$ are true in $\mathcal{I}$. The first formula expresses that there is no node in the transition system where $\omega(r_1)$ holds and $\omega(r_2)$ does not hold. This is equivalent to $\omega(r_1) \Rightarrow \omega(r_2)$ and $[\![r_1]\!]^K \subseteq [\![r_2]\!]^K$ since r_1 and r_2 are basic roles. Thus, we obtain $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$ and $\mathcal{I} \models r_1 \sqsubseteq r_2$. Similar justifications as above can be worked out to arrive at $\mathcal{I} \models C_1 \sqsubseteq C_2$ since C_1 and C_2 are basic concepts. □

In the following, for legibility, we denote $\Phi(\mathcal{S}, q, q') = \eta(\mathcal{S}) \wedge \mathcal{A}(q) \wedge \neg\mathcal{A}(q', m) \wedge \varphi_r$.

Theorem 4 (Soundness and Completeness)**.** *Given queries q, q', and a set of RDF schema axioms $\mathcal{S}$, $q \sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$ if and only if $\Phi(\mathcal{S}, q, q')$ is unsatisfiable.*

Proof. (sketch) *Soundness:* $\Phi(\mathcal{S}, q, q')$ unsatisifiable implies that $q \sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$. We show the contrapositive, if $q \not\sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$, then $\Phi(\mathcal{S}, q, q')$ is satisfiable, holds. One can verify that every model G of $\mathcal{S}$ in which there is at least one triple satisfying q but not q' can be turned into a transition system model for $\Phi(\mathcal{S}, q, q')$.
Completeness: $\Phi(\mathcal{S}, q, q')$ satisfiable implies $q_1 \not\sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q_2$. Assume that there exists a restricted transition system K that satisfies $\Phi(\mathcal{S}, q, q')$. This entails that, $[\![\varphi_r]\!]^K \neq \emptyset$ (cf. Proposition 1). Now, from $K = (S, R, L)$ we need to construct an RDF graph G that is model of $\mathcal{S}$ such that $q \not\sqsubseteq^{\mathcal{S}}_{\text{rdfs}} q'$ holds:

- for every RDFS concept C in the schema, $\{(s, \texttt{type}, C) \mid \forall s', s'' \in S \wedge t \in S'.(s', t) \in R(s) \wedge (t, s'') \in R(p) \wedge (t, s) \in R(o) \wedge s \in [\![C]\!]^K\}$.
- for each RDFS property P in the schema, $\{(s, P, s') \in G \mid \forall t \in [\![P]\!]^K \wedge t' \in S.(s, t') \in R(s) \wedge (t', t) \in R(p) \wedge (t', s') \in R(o)\}$,
- add every schema axiom to G and for each triple $t_i \in q$, add t_i to G.

Since every RDF graph entails its schema graph, we obtain that G is a model of $\mathcal{S}$. Thus, it remains to show that $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. From our assumption, one anticipates $[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q')]\!]^K \neq \emptyset$ which implies $[\![\mathcal{A}(q)]\!]^K \neq \emptyset$ and $[\![\mathcal{A}(q', m)]\!]^K = \emptyset$.

$$\begin{aligned}[\![\mathcal{A}(q) \wedge \neg\mathcal{A}(q')]\!]^K \neq \emptyset \quad &\Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\neg\mathcal{A}(q', m)]\!]^K \neq \emptyset \\ &\Rightarrow [\![\mathcal{A}(q)]\!]^K \neq \emptyset \text{ and } [\![\mathcal{A}(q', m)]\!]^K = \emptyset\end{aligned}$$

Note here that, if a formula φ is satisfiable in a restricted transition system K, then $[\![\varphi]\!]^K = S$. Further, it is holds that $[\![q]\!]_G \neq \emptyset$ and $[\![q']\!]_G = \emptyset$ because G contains all those triples that satisfy q and not q'. Therefore, we get $[\![q]\!]_G \not\subseteq [\![q']\!]_G$. Since cycles in queries can be expressed by a formula in a μ-calculus extended with nominals and inverse, the constraints expressed by $\neg\mathcal{A}(q', m)$ are satisfied in a transition system containing cycles. □

Theorem 5 (PSPARQL query containment)**.** *Given two PSPARQL queries q_1 and q_2, $q_1 \sqsubseteq q_2$ iff $\mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2) \wedge \varphi_r$ is unsatisfiable.*

Proof. The proof follows from the proof of Theorem 3. □

5.4 Complexity

Due to duplication in the encoding of the right hand side query q', the size of $|\mathcal{A}(q', m)|$ is exponential in terms of the non-distinguished variables that appear in cycles in the query. Thus, we obtain a 2ExpTime upper bound for containment independent of the approaches. That is, the complexity bound applies to all the approaches. As pointed out in [5], the problem is solvable in ExpTime if there is no cycle in the query on the right hand side. In this case, this complexity is a lower bound due to the complexity of satisfiability in μ-calculus.

Proposition 2. *SPARQL query containment under the RDFS entailment can be solved in a time of $2^{\mathcal{O}(n)}$, where n is the size of the encoding.*

All the three approaches have the same complexity bound, the difference lies on their extensibility. While encoding the RDFS semantics (§**??**) and query rewriting (§**??**) approaches are tied to the schema language which makes it harder for easy extension, the schema encoding approach (§5.3) can be extended to use a more expressive schema language than RDFS. For instance, we can extend the schema language to $\mathcal{SH}$ where a concept C can be a bottom concept ($\top$), an atomic concept A, or a complex concept $\neg C$ or $C \sqcap D$. A role r is an atomic role. An $\mathcal{SH}$ TBox consists of concept inclusion, role inclusion and role transitivity axioms [16]. Role inclusion and transitivity axioms can be encoded in the same way as it is done in Definition 13. The encoding of concept inclusion axioms is slightly different, thus, we extend η as follows:

$$\begin{aligned}\eta((C, \mathtt{sc}, D)) &= \nu X.\ (\omega(C) \Rightarrow \omega(D)) \wedge nu(X) \\ \omega(\bot) &= \bot \\ \omega(A) &= A \\ \omega(\neg C) &= \neg\omega(C) \\ \omega(C \sqcap D) &= \omega(C) \wedge \omega(D)\end{aligned}$$

We can expand the proof of Theorem 2, to prove the correctness of this reduction. And thus, retaining the double exponential upper bound. Beyond this, we can even extend $\mathcal{SH}$ to the fragments of $\mathcal{SROIQ}$ [15]. More specifically, the fragments without number restrictions. The expressiveness of the schema language is limited as such due to the expressive power of the logic used for the encoding: μ-calculus with nominals and converse becomes undecidable when extended with graded modalities [4].

6 Conclusion

In this work, we have presented a translation of RDF graphs into labeled transition systems over which μ-calculus formulas are interpreted. We also have provided functions to produce the encodings of queries, inference rules and schema as formulas. Henceforth, query containment under RDFS entailment is reduced to formula satisfiability test in the μ-calculus. We introduced three approaches to achieve this, namely (1) encoding the RDFS semantics, (2) query rewriting, and (3) encoding the schema. Unlike (1) and (2), the third approach can be extended for a more expressive schema language as shown in §5.4, while maintaining a double exponential upper bound complexity. The power of the logic and our encoding allows for taking advantage of more expressive schema language. For instance, a good candidate could be the description logic $\mathcal{SROIQ}$ [15] underlying OWL 2.

In the future, we plan to investigate the optimality of the upper bound considering a more expressive schema language than RDF schema. Additionally, we plan to study containment of path queries with counting quantifiers (SPARQL 1.1 property hierarchies) using the graded flavour of the μ-calculus [4].

References

[1] Faisal Alkhateeb, Jean-François Baget, and Jérôme Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Semantics*, 7(2):57–73, 2009.

[2] Renzo Angles and Claudio Gutierrez. The Expressive Power of SPARQL. *The Semantic Web-ISWC 2008*, pages 114–129, 2008.

[3] Pablo Barceló, Carlos Hurtado, Leonid Libkin, and Peter Wood. Expressive languages for path queries over graph-structured data. In *PODS'10*, pages 3–14. ACM, 2010.

[4] Piero A. Bonatti, Carsten Lutz, Aniello Murano, and Moshe Y. Vardi. The Complexity of Enriched μ-calculi. *Automata, Languages and Programming*, pages 540–551, 2006.

[5] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive Query Containment and Answering under Description Logics Constraints. *ACM Trans. on Computational Logic*, 9(3):22.1–22.31, 2008.

[6] Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Containment of regular path queries under description logic constraints. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, 2011.

[7] Diego Calvanese and Riccardo Rosati. Answering Recursive Queries under Keys and Foreign Keys is Undecidable. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*, volume 79, pages 3–14, 2003.

[8] Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[9] A. Chebotko, S. Lu, H.M. Jamil, and F. Fotouhi. Semantics preserving sparql-to-sql query translation for optional graph patterns. Technical report, Technical Report TR-DB-052006-CLJF, 2006.

[10] Melisachew Wudage Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. PSPARQL query containment. In *DBPL'11*, August 2011.

[11] R. Cyganiak. A relational algebra for SPARQL. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, 2005.

[12] Pierre Genevès, Nabil Layaïda, and Alan Schmitt. Efficient Static Analysis of XML Paths and Types. In *PLDI '07*, pages 342–351, New York, NY, USA, 2007. ACM.

[13] Birte Glimm. Using SPARQL with RDFS and OWL entailment. *Reasoning Web. Semantic Technologies for the Web of Data*, pages 137–201, 2011.

[14] Patrick Hayes. RDF Semantics. W3C Recommendation, 2004.

[15] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In *Proc. of KR 2006*, pages 57–67, 2006.

[16] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Logic for Programming and Automated Reasoning*, pages 161–180. Springer, 1999.

[17] Yannis E. Ioannidis. Query Optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

[18] Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comp. Sci.*, 27:333–354, 1983.

[19] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.

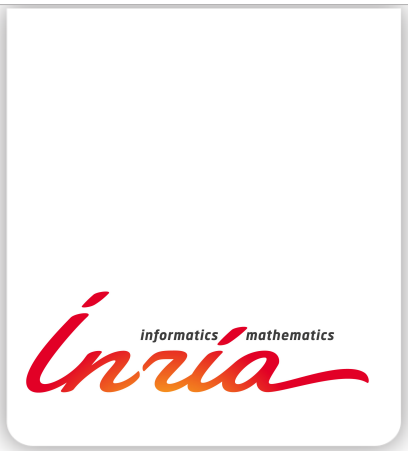
[20] Axel Polleres. From SPARQL to rules (and back). In *WWW '07*, pages 787–796, 2007.

[21] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Rec., 2008.

[22] Yoshinori Tanabe, Koichi Takahashi, and Masami Hagiya. A Decision Procedure for Alternation-Free Modal μ-calculi. In *Advances in Modal Logic*, pages 341–362, 2008.

[23] Yoshinori Tanabe, Koichi Takahashi, Mitsuharu Yamamoto, Akihiko Tozawa, and Masami Hagiya. A Decision Procedure for the Alternation-Free Two-Way Modal μ-calculus. In *TABLEAUX*, pages 277–291, 2005.

Contents



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399